Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER Computers and Mathematics with Applications 48 (2004) 275–289
An International Journal computers & mathematics with applications
www.elsevier.com/locate/camwa

Mapping Matrix Multiplication Algorithm onto Fault-Tolerant Systolic Array

N. M. STOJANOVIĆ AND E. I. MILOVANOVIĆ
Faculty of Electronic Engineering
Beogradska 14, P.O. Box 73, 18000 Niš, Serbia

I. STOJMENOVIĆ
Computer Science Department, University of Ottawa
Ottawa, Ontario K1N 6N5, Canada

I. Ž. MILOVANOVIĆ AND T. I. TOKIĆ
Faculty of Electronic Engineering
Beogradska 14, P.O. Box 73, 18000 Niš, Serbia

(Received March 2001; revised and accepted December 2002)

Abstract—An approach to design fault-tolerant hexagonal systolic array (SA) for multiplication of rectangular matrices is described. The approach comprises three steps. First, redundancies are introduced at the computational level by deriving three equivalent algorithms but with disjoint index spaces. Second, we perform the accommodation of index spaces to the projection direction to obtain hexagonal SA with optimal number of processing elements (PE) for a given problem size. Finally, we perform mapping of the accommodated index spaces into fault-tolerant systolic array using valid transformation matrix. As a result, we obtain SA with optimal number of PEs which performs fault-tolerant matrix multiplication. In the case of square matrices of order $N \times N$, this array comprises $N^2 + 2N$ PEs with active computation time $t_c = 5N - 4$ time units. Fault tolerance is achieved through triplicated computation of the same problem instance and majority voting. We have proposed two hardware solutions for the voting process: one when voting is performed at the end of the computation, i.e., at the output of the SA, and the other where voting is performed after each computational step. With the proposed method, any single transient or permanent fault can be detected and corrected. Experimental results show that with the proposed schemes a lot of multiple error patterns can be tolerated, also.

Keywords—Matrix multiplication, Fault-tolerance, Systolic arrays.

1. INTRODUCTION

Matrix multiplication plays a central role in numerical linear algebra, since we met it in almost all numerical algorithms, including matrix inversion, eigenvalue computation, numerical solution of the PDE, as well as in many technical problems including digital signal processing, circuit simulation, digital control, etc. Matrix multiplication is a very regular computation and lends itself well to parallel implementation. Regular structures such as systolic arrays (SA) are well suited for matrix multiplication and are also amenable to VLSI/WSI implementation because of simple and regular design, and nearest neighbor communications.

0898-1221/04/$ - see front matter doi:10.1016/j.camwa.2002.12.016

Fault tolerance has become a crucial design requirement for VLSI/WSI array processors. Fault tolerance can be achieved through some form of redundancy, i.e., either information (ABFT) [1,2], space and/or time [3–5], and by reconfiguration.

In this paper, we present a systematic approach to design a fault-tolerant VLSI/WSI SA for multiplication of matrix A of order $N_1 \times N_3$ by matrix B of order $N_3 \times N_2$ to obtain matrix C of order $N_1 \times N_2$. The method is based on fault-tolerant mapping theory which is developed from space-time mapping technique [6–10], and the theory on concurrent error detection using space/time redundancy [4,5,11]. Fault tolerance is achieved through triplicated computation of the same problem instance and majority voting. Our mapping algorithm to obtain fault-tolerant systolic array is based on composition of two linear transformations (H, T), contrary to the approach proposed in [5], which uses only valid transformation matrix T. Linear transformation H accommodates index space of the matrix multiplication algorithm to the projection direction μ. This accommodation enables us to obtain SA with optimal number of processing elements (PE) for a given problem size. Optimal SA is obtained when valid transformation T is applied on the accommodated index space. The total number of PEs, n_p, and the total computation time, t_c, are two major performance measures. Therefore, we take their product $AT = n_p \times t_c$ (space-time complexity) as the indicator of the SA optimality. The fault tolerant SA for matrix multiplication obtained by the proposed method has $AT = N_3(\min\{N_1, N_2\} + 2)(3\max\{N_1, N_2\} + \min\{N_1, N_2\} + N_3 - 4)$ compared to $AT = (N_1(N_2 - 1) + N_2(N_3 + 1) + N_3(N_1 + 1) - 1)(N_1 + N_2 + N_3 - 2)$ obtained in [5]. When $N_1 = N_2 = N_3 = N$ we, respectively, obtain $AT = (N^2 + 2N)(5N - 4) = O(5N^3)$, compared to $AT = (3N^2 + N - 1)(3N - 2) = O(9N^3)$.

2. BACKGROUND

Let $A = (a_{ik})$ and $B = (b_{kj})$ be two matrices of order $N_1 \times N_3$ and $N_3 \times N_2$, respectively. To find their product, $C = A \cdot B$, the following recurrence relation can be used:

$$c_{ij}^{(k)} := c_{ij}^{(k-1)} + a_{ik} b_{kj}, \qquad k = 1, 2, \ldots, N_3, \tag{2.1}$$

for all $i = 1, 2, \ldots, N_1$, and $j = 1, 2, \ldots, N_2$, where $c_{ij}^{N_3} = c_{ij}$. The systolic algorithm that computes $C = A \cdot B$ according to (2.1) has the following form.

ALGORITHM_1.

```
for k := 1 to N3 do
  for j := 1 to N2 do
    for i := 1 to N1 do
      a(i,j,k) := a(i,j-1,k);
      b(i,j,k) := b(i-1,j,k);
      c(i,j,k) := c(i,j,k-1) + a(i,j,k) * b(i,j,k);
endfor{i,j,k,};
```

where $a(i, 0, k) \equiv a_{ik}$, $b(0, j, k) \equiv b_{kj}$ and $c(i, j, k) \equiv c_{ij}^{(k)}$. We will call this algorithm the basic one.

Now, we will briefly explain the main steps for synthesizing 2D planar arrays that implement matrix multiplication according to Algorithm_1. The computational structure of the Algorithm_1 is determined by the inner computation space (see, for example, [12])

$$P_{\text{int}} = \{(i, j, k) \mid 1 \le i \le N_1,\ 1 \le j \le N_2,\ 1 \le k \le N_3\}, \tag{2.2}$$

where data are used or computed, and a dependency matrix

$$D = [\vec{e}_b^{\,3} \vec{e}_a^{\,3} \vec{e}_c^{\,3}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.3}$$

The space of initial computations, $P_{\text{in}} = \{P_{\text{in}}(a) \cup P_{\text{in}}(b) \cup P_{\text{in}}(c)\}$, is defined as

$$P_{\text{in}}(a) = \{(i,0,k) \mid 1 \le i \le N_1,\ 1 \le k \le N_3\},$$
$$P_{\text{in}}(b) = \{(0,j,k) \mid 1 \le j \le N_2,\ 1 \le k \le N_3\},$$
$$P_{\text{in}}(c) = \{(i,j,0) \mid 1 \le i \le N_1,\ 1 \le j \le N_2\}.$$

An important performance parameter of the designed SA is the pipeline period α, which is the time interval in clock units between two successive activities of a processor. $\alpha = 1$ means that a processor is busy in every clock, $\alpha = 2$ means the processor is activated in every other clock, etc. Here we are interested in hexagonal arrays with pipeline period $\alpha = 3$, since they can be used to achieve fault-tolerant computing.

Hexagonal SA with the pipeline period $\alpha = 3$ is obtained by mapping computational structure (D, P_{int}) along the direction $\vec{\mu} = [1\,1\,1]^{\top}$ using one of the transformation matrices of the form

$$T = \begin{bmatrix} \vec{\Pi} \\ S \end{bmatrix} = \begin{bmatrix} \vec{\Pi} \\ S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix}. \tag{2.4}$$

The mapping is performed as

$$T : (D, P_{\text{int}}) \longmapsto (\Delta, \bar{P}_{\text{int}}), \tag{2.5}$$

where $\bar{P}_{\text{int}} = \{(t,x,y)\}$. Here t represents the time instance when the computation is performed, while $(x,y) = \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ denotes the coordinates of the PE where the computation is taking place. Both refer to the index point (i,j,k). The communication links between the PEs are implemented along the projection of data dependency vectors; i.e.,

$$\Delta_s = S \cdot D = \left[\vec{e}_b^{\,2} \vec{e}_a^{\,2} \vec{e}_c^{\,2}\right]. \tag{2.6}$$

Standard synthesis procedure (see, for example, [14,15]) does not take into account some features of the computations in Algorithm_1 that can be used to minimize space parameters of the SA.

As we have mentioned, one of the features of this array is pipeline period $\alpha = 3$. Other planar arrays have $1 \le \alpha \le 2$. A systolic array is not fully utilized if $\alpha > 1$. One way to increase the SA utilization is to execute two or more problem instances simultaneously (see, for example, [10]). Another is to perform replicated computations of the same problem instance to achieve fault tolerance (see, for example, [3,5,11]).

In the error masking approach, which is known as N-tuple modular redundancy, N copies (N odd) of a module and majority voter are used to mask the error from a failed module. At least three modules are necessary in a voting system that is typically called triple modular redundancy (TMR). It seems that at least 200 percent hardware overhead for fault tolerance is needed. In practice, it needs to put triplicate computations to the voter, and then, gets a correct result. The triplicated computations may be computed in different PEs and/or different time using space-shift, time-shift, or space-time-shift [11]. If the replicated computations are performed simultaneously by different PEs, it is a space-shift scheme. If the replicated computations are computed by the same PE at different times it is a time-shift scheme. If the replicated computations are computed by different PEs at different times, it is a space-time-shift scheme.

A systolic array with pipeline period $\alpha = 3$ can perform an original algorithm and two redundant algorithms derived from the original one, concurrently. These redundant computations can be performed by the idle PEs at idle clock cycles. In the text that follows, we will briefly describe the idea employed in [5] to design a fault-tolerant hexagonal SA for matrix multiplication. By the proposed method, any single error at any given time can be detected and corrected.

Zhang *et al.* [5] have observed that an additional two algorithms equivalent to Algorithm_1 can be derived by translating the space $P = P_{\text{in}} \cup P_{\text{int}}$ along the vectors $\vec{d} = [-(2/3)\ (1/3)\ (1/3)]^{\top}$ and $\vec{d} = [-(4/3)\ (2/3)\ (2/3)]^{\top}$. The common description of all three algorithms is as follows.

ALGORITHMS_2.

```
for r := 0 to 2 do
 for k := 1 to N3 do
  for j := 1 to N2 do
   for i := 1 to N1 do
    a(i − 2r/3, j + r/3, k + r/3) := a(i − 2r/3, j + r/3 − 1, k + r/3);
    b(i − 2r/3, j + r/3, k + r/3) := b(i − 2r/3 − 1, j + r/3, k + r/3);
    c(i − 2r/3, j + r/3, k + r/3) := c(i − 2r/3, j + r/3, k + r/3 − 1)+
     +a(i − 2r/3, j + r/3, k + r/3) * b(i − 2r/3, j + r/3, k + r/3);
   endfor{i, j, k, r};
```

For $r = 0, 1, 2$, three equivalent algorithms with disjoint index spaces are obtained. The initial values in Algorithms_2 are given by $a(i-2r/3,\ r/3,\ k+r/3) \equiv a_{ik}$, $b(-2r/3,\ j+r/3,\ k+r/3) \equiv b_{kj}$, and $c(i - 2r/3,\ j + r/3,\ r/3) \equiv 0$. The final results of Algorithms_2 are $c(i - 2r/3,\ j + r/3,\ N_3 + r/3) \equiv c_{ij}$.

The inner computation space of the above three algorithms are

$$P_{\text{int}}(r) = \left\{ \left(i - \frac{2r}{3},\ j + \frac{r}{3},\ k + \frac{r}{3} \right) \;\middle|\; 0 \le r \le 2,\ 1 \le i \le N_1,\ 1 \le j \le N_2,\ 1 \le k \le N_3 \right\}.$$

Using valid transformation matrix [5]

$$T = \left[\frac{\vec{\Pi}}{\mathrm{S}} \right] = \left[\begin{array}{ccc} 1 & 1 & 1 \\ \hline -1 & 1 & 0 \\ 0 & -1 & 1 \end{array} \right], \tag{2.7}$$

the computational structure $(D, P_{\text{int}}(r))$ is mapped into fault-tolerant SA, i.e.,

$$T : (D, P_{\text{int}}(r)) \longmapsto \left(\Delta, \bar{P}_{\text{int}}(r)\right).$$

Note that several valid transformations T that map $(D, P_{\text{int}}(r))$ into $(\Delta, \bar{P}_{\text{int}}(r))$ can be generated. For T defined by (2.7), the (x, y) positions of PEs in the SA are given by

$$\begin{bmatrix} x \\ y \end{bmatrix} = S \cdot \vec{p} = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} i - \dfrac{2r}{3} \\ j + \dfrac{r}{3} \\ k + \dfrac{r}{3} \end{bmatrix} = \begin{bmatrix} j - i + r \\ k - j \end{bmatrix}, \qquad (\vec{p} \in P_{\text{int}}(r)), \tag{2.8}$$

while time schedule, t, is determined according to

$$t = t\left(i - \frac{2r}{3},\ j + \frac{r}{3},\ k + \frac{r}{3} \right) = i + j + k. \tag{2.9}$$

The communication links between the PEs are implemented along the projections of data dependency vectors; i.e.,

$$\Delta_s = [\vec{e}_b^{\,2} \vec{e}_a^{\,2} \vec{e}_c^{\,2}] = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \cdot D = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}. \tag{2.10}$$

The array obtained according to (2.8)–(2.10) has the following features:

- number of PEs: $n_p = N_1(N_2 - 1) + N_2(N_3 + 1) + N_3(N_1 + 1) - 1$,
- computation time: $t_c = N_1 + N_2 + N_3 - 2$,
- space-time complexity: $AT = n_p \times t_c = [N_1(N_2 - 1) + N_2(N_3 + 1) + N_3(N_1 + 1) - 1][N_1 + N_2 + N_3 - 2]$.

For the case $N_1 = N_2 = N_3 = N$, the following is obtained [4,5]:

$$\begin{aligned} n_p &= 3N^2 + N - 1, \qquad t_c = 3N - 2, \\ AT &= \left(3N^2 + N - 1\right)(3N - 2) = O\left(9N^3\right). \end{aligned} \tag{2.11}$$

This array, as well as the original one without fault-tolerant capability, has a large number of PEs. In the next section, we will describe a modification of the synthesis procedure that generates hexagonal arrays with optimal number of PEs with respect to a problem size.

The corresponding SA that implements fault-tolerant matrix multiplication for $N_1 = N_2 = N_3 = 3$ is shown in Figure 1. This array is referred to as optimal fault-tolerant SA with respect to a space-time complexity in [5].

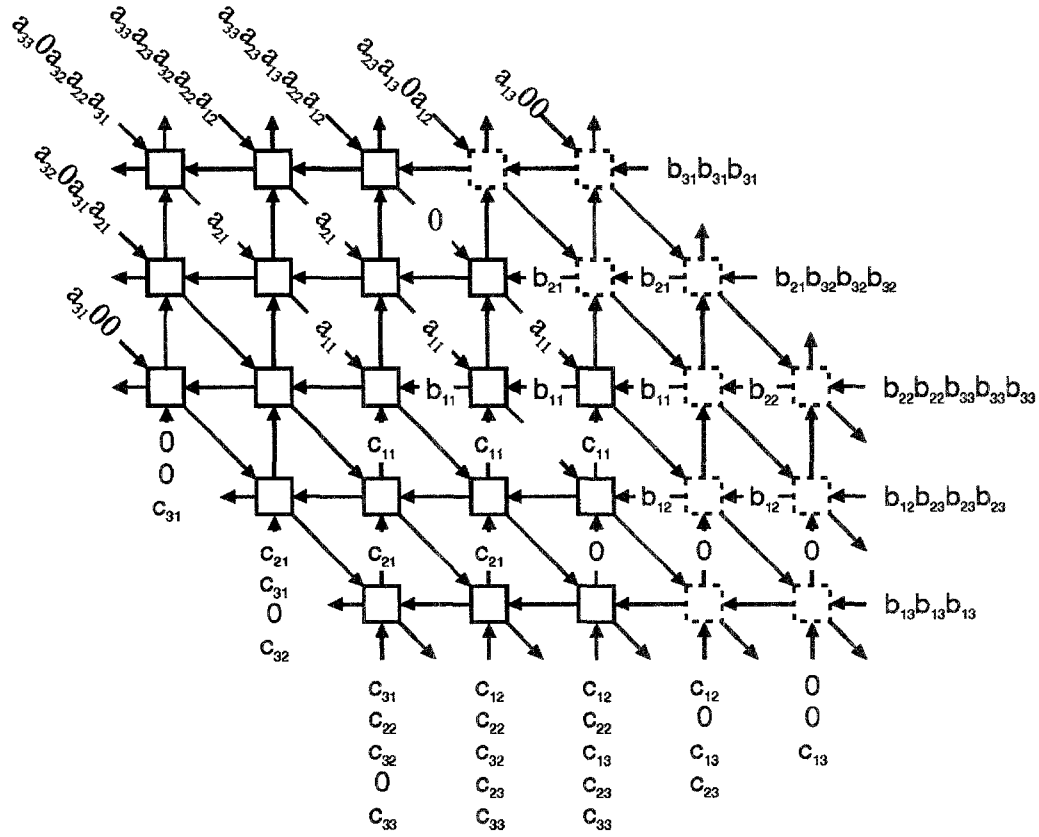

Figure 1. Data flow in the SA during fault-tolerant matrix multiplication for $N_1 = N_2 = N_3 = 3$, obtained in [5].

3. MODIFICATION OF THE SYNTHESIS PROCEDURE

In what follows, we will briefly describe the design procedure that enables us to obtain SAs with an optimal number of PEs without fault-tolerance capabilities. Then, we will use it to obtain the SA with optimal number of PEs that performs fault-tolerant matrix multiplication.

The inner computation space P_{int} of Algorithm_1 is a cubic lattice where each node has integer coordinates in the corresponding $(\vec{i}, \vec{j}, \vec{k})$ Cartesian space. The orthogonal projection of this lattice on the $(\vec{i}\,\vec{j})$-plane has N_1N_2 nodes, on the $(\vec{i}\,\vec{k})$-plane N_1N_3 nodes, and on the $(\vec{j}\,\vec{k})$-plane N_2N_3 nodes. Since the projection on $(\vec{i}\,\vec{k})$-plane cannot be used to obtain a systolic array that performs fault-tolerant computation, we will not take it into consideration.

Depending on the relation between N_1 and N_2, we can choose the projection on either the $(\vec{i}\,\vec{k})$-plane or the $(\vec{j}\,\vec{k})$-plane to obtain the image with minimal number of nodes, namely the one

which consists of

$$n_p = N_3 \min\{N_1, N_2\} \tag{3.1}$$

nodes. It is well known that nonorthogonal projections give images with larger number of nodes than (3.1). Consequently, the same is valid for the image obtained by the projection vector $\vec{m} = [1\,1\,1]^\top$. Therefore, we are interested to find out a mapping H which maps the inner computation space P_{int} into an equivalent one, denoted as P^*_{int}, i.e.,

$$H : P_{\text{int}} \longmapsto P^*_{\text{int}}, \tag{3.2}$$

such that after applying T on (D, P^*_{int}), i.e.,

$$T : (D, P^*_{\text{int}}) \longmapsto (\Delta, \bar{P}_{\text{int}}) \tag{3.3}$$

the obtained image, i.e., the corresponding SA, contains the optimal number of PEs with respect to a problem size, namely the one determined by (3.1). Of course, this transformation must not violate the correctness of the computation.

Let us consider the mapping $H, = (F, G)$, defined as

$$F = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \qquad G = \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix}, \tag{3.4}$$

and

$$F = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \qquad G = \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix}. \tag{3.5}$$

Accordingly, mapping (3.2) can be rewritten as

$$[u \quad v \quad w]^\top = F \cdot [i \quad j \quad k]^\top + G, \tag{3.6}$$

for each $i = 1, \ldots, N_1$, $j = 1, \ldots, N_2$, and $k = 1, \ldots, N_3$. Let us observe that H maps projection vector $\vec{\mu} = [1\,1\,1]^\top$ onto itself, i.e., $H : \vec{\mu} \mapsto \vec{\mu}$, regardless of whether H is defined as (3.4) or (3.5). Actually, H performs an accommodation of index space to the projection direction $\vec{\mu}$ by skewing index space.

Suppose, for example, that H is defined by (3.4). According to (3.6), we obtain that

$$P^*_{\text{int}} = \left\{ [i \quad i+j-1 \quad i+k-1]^\top \mid 1 \le i \le N_1,\ 1 \le j \le N_2,\ 1 \le k \le N_3 \right\}.$$

Now, consider the subset $P^*_{\text{int}}(j,k) \subset P^*_{\text{int}}$, defined as

$$P^*_{\text{int}}(j,k) = \left\{ [i \quad i+j-1 \quad i+k-1]^\top \mid 1 \le i \le N_1 \right\}$$

for arbitrary j and k, $1 \le j \le N_2$, $1 \le k \le N_3$. Since for all points with position vector $\vec{p}^*_1 = [i \quad i+j-1 \quad i+k-1]^\top$ and $\vec{p}^*_2 = [i+1 \quad i+j \quad i+k]^\top$ from $P^*_{\text{int}}(j,k)$, hold $\vec{p}^*_2 - \vec{p}^*_1 = [1\,1\,1]^\top = \vec{\mu}$, we conclude that all index points from the space $P^*_{\text{int}}(j,k)$ are placed on the line with direction $\vec{\mu}$. This means that mapping (3.2) maps all index points from the subset $P^*_{\text{int}}(j,k)$ into one point in the projection plane. Since there are totally $N_2 N_3$ different subsets $P^*_{\text{int}}(j,k)$, $1 \le j \le N_2$, $1 \le k \le N_3$, we conclude that the image $\bar{P}_{\text{int}}$ of P^*_{int} has $N_2 N_3$ points. Actually, we have proved that with the composition of two mappings, H defined by (3.4) and T defined by (3.3), a systolic array with $n_p = N_2 N_3$ PEs is obtained.

Similarly, if composition of mapping H, defined by (3.4), and T defined by (3.3), is applied, a systolic array with $n_p = N_1 N_3$ PEs will be obtained.

Accordingly, depending on the relation between N_1 and N_2 (i.e., $N_1 > N_2$ or $N_1 < N_2$) we choose mapping H defined by (3.4) or (3.5) to obtain the SA with number of PEs determined by (3.1).

To see that correctness of the computation is preserved it is enough to see that, for example, according to H defined by (3.4) an algorithm equivalent to Algorithm_1 can be derived. The inner computation space of this new algorithm is P^*_{int}. The algorithm has the following form.

ALGORITHM_3.

```
for k := 1 to N3 do
 for j := 1 to N2 do
  for i := 1 to N1 do
   a(i, i+j-1, i+k-1) := a(i, i+j-2, i+k-1);
   b(i, i+j-1, i+k-1) := b(i-1, i+j-1, i+k-1);
   c(i, i+j-1, i+k-1) := c(i, i+j-1, i+k-2)+a(i, i+j-1, i+k-1)*b(i, i+j-1, i+k-1);
endfor{i, j, k,};
```

whereas for initial values the following is valid: $a(i,0,k) \equiv a(i,0,k+N_3) \equiv a_{ik}$, $b(0,i,k) \equiv b(0,j+N_2,k) \equiv b(0,j,k+N_3) \equiv b_{kj}$. Here we use the fact that for some fixed i the computations in Algorithm_3 can be performed over arbitrary permutation of index variables j and k (see, for example, [13]).

4. DESIGNING FAULT-TOLERANT SA FOR MATRIX MULTIPLICATION

It is not difficult to conclude that index variables i, j, and k were equally treated in Algorithms_2. In our approach, as we have explained in the previous section, we make a difference whether $N_1 \geq N_2$ or $N_1 < N_2$ in order to perform space optimization. Therefore, instead of Algorithms_2 we will consider other three equivalent algorithms depending on the relation between N_1 and N_2.

When $N_1 \geq N_2$ we use the following three algorithms to design fault-tolerant SA with optimal number of PEs.

ALGORITHMS_4.

```
for r := 0 to 2 do
 for k := 1 to N3 do
  for j := 1 to N2 do
   for i := 1 to N1 do
    a(i+r/3, j-r, k) := a(i+r/3, j-r-1, k);
    b(i+r/3, j-r, k) := b(i+r/3-1, j-r, k);
    c(i+r/3, j-r, k) := c(i+r/3, j-r, k-1) + a(i+r/3, j-r, k) * b(i+r/3, j-r, k);
   endfor{i, j, k, r};
```

with the following initial values: $a(i+r/3,-r,k) \equiv a_{ik}$, $b(r/3,j-r,k) \equiv b_{kj}$, and $c(i+r/3,j-r,0) \equiv 0$. The final results of Algorithms_4 are $c(i+r/3,j-r,N_3) \equiv c_{ij}$. The inner computation spaces of Algorithms_4 are given by

$$P_{\text{int}}(r) = \left\{ \left(i + \frac{r}{3}, j-r, k \right) \;\middle|\; 0 \leq r \leq 2,\ 1 \leq i \leq N_1,\ 1 \leq j \leq N_2,\ 1 \leq k \leq N_3 \right\}.$$

The data dependency matrix is the same as for Algorithms_1, i.e., as one given by (2.3). In the case $N_1 \geq N_2$ we take H defined by (3.4) and T defined as

$$T = \begin{bmatrix} \vec{\Pi} \\ S \end{bmatrix} = \left[\begin{array}{ccc} 1 & 1 & 1 \\ \hline -1 & 1 & 0 \\ -1 & 0 & 1 \end{array}\right].$$

Note that transformation matrix T is not the same as one given by (2.7). Namely, under the conditions defined in [9], the transformation T given by (2.7) is no more valid, since it would generate SA with bad spatial features.

Now, according to (3.2) and (3.3), i.e., according to mappings $H : P_{\text{int}}(r) \mapsto P^*_{\text{int}}(r)$ and $T : (D, P^*_{\text{int}}) \mapsto (\Delta, \bar{P}_{\text{int}})$, we obtain that (x, y) positions of the PEs in the hexagonal SA are obtained as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} i + \dfrac{r}{3} \\ i + j - \dfrac{2r}{3} - 1 \\ i + k + \dfrac{r}{3} - 1 \end{bmatrix} = \begin{bmatrix} j - r - 1 \\ k - 1 \end{bmatrix}$$

for each $0 \le r \le 2$, $1 \le i \le N_1$, $1 \le j \le N_2$, $1 \le k \le N_3$. The time schedule of the data item indexed by (i, j, k) is given by

$$t = t\left(i + \frac{r}{3}, i + j - \frac{2r}{3} - 1, i + k + \frac{r}{3} - 1\right) = 3i + j + k - 2.$$

Note that for input data items, we assume the following periodicity:

$$a\left(i + \frac{r}{3} + N_1, j - \frac{2r}{3}, k + \frac{r}{3}\right) \equiv a\left(i + \frac{r}{3}, j - \frac{2r}{3}, k + \frac{r}{3} + N_3\right) \equiv a_{ik},$$
$$b\left(i + \frac{r}{3}, j - \frac{2r}{3} + N_2, k + \frac{r}{3}\right) \equiv b\left(i + \frac{r}{3}, j - \frac{2r}{3}, k + \frac{r}{3} + N_3\right) \equiv b_{kj}.$$

The initial (x, y) positions of input data items are determined from

$$a\left(i + \frac{r}{3}, 0, i + k + \frac{r}{3} - 1\right) \longmapsto \begin{bmatrix} x \\ y \end{bmatrix}_a = \begin{bmatrix} 3 - 3i - k - r \\ k - 1 \end{bmatrix},$$
$$b\left(0, i + j - 1 - \frac{2r}{3}, i + k + \frac{r}{3} - 1\right) \longmapsto \begin{bmatrix} x \\ y \end{bmatrix}_b = \begin{bmatrix} 3i + 2j + k - r - 5 \\ 3i + j + 2k - 5 \end{bmatrix},$$
$$c\left(i + \frac{r}{3}, i + j - \frac{2r}{3} - 1, 0\right) \longmapsto \begin{bmatrix} x \\ y \end{bmatrix}_c = \begin{bmatrix} j - r - 1 \\ 3 - 3i - j \end{bmatrix},$$

for each $0 \le r \le 2$, $1 \le i \le N_1$, $1 \le j \le N_2$, $1 \le k \le N_3$. The obtained SA has the following features:

$$\begin{aligned} n_p &= N_3(N_2 + 2), \\ t_c &= 3N_1 + N_2 + N_3 - 4, \\ AT &= N_3(N_2 + 2)(3N_1 + N_2 + N_3 - 4). \end{aligned} \tag{4.1}$$

For $N_1 = N_2 = N_3 = N$, we have

$$n_p = N(N + 2), \qquad t_c = 5N - 4, \qquad AT = N(N + 2)(5N - 4) = O\left(5N^3\right). \tag{4.2}$$

When $N_1 < N_2$, accommodation of $P_{\text{int}}(r)$ is performed over index variable j. Now, we start from the following three algorithms.

ALGORITHMS_5.

```
for r := 0 to 2 do
 for k := 1 to N3 do
  for j := 1 to N2 do
   for i := 1 to N1 do
    a(i − r, j + r/3, k) := a(i − r, j + r/3 − 1, k);
    b(i − r, j + r/3, k) := b(i − r − 1, j + r/3, k);
    c(i − r, j + r/3, k) := c(i − r, j + r/3, k − 1) + a(i − r, j + r/3, k) * b(i − r, j + r/3, k);
   endfor{i, j, k, r};
```

with the following initial values: $a(i-r, r/3, k) \equiv a_{ik}$, $b(-r, j+r/3, k) \equiv b_{kj}$, and $c(i-r, j+r/3, 0) = 0$. The final results are $c(i-r, j+r/3, k+N_3) \equiv c_{ij}$.

The inner computation spaces of Algorithms_5 are given by

$$P_{\text{int}}(r) = \left\{ \left(i-r, j+\frac{r}{3}, k \right) \;\middle|\; 0 \le r \le 2,\ 1 \le i \le N_1,\ 1 \le j \le N_2,\ 1 \le k \le N_3 \right\}.$$

The data dependency matrix is the same as the one given by (2.2). In order to accommodate $P_{\text{int}}(r)$ to the direction $\vec{\mu} = [1\,1\,1]^{\top}$ over index variable j, we take H defined by (3.5). For transformation T we choose

$$T = \begin{bmatrix} \vec{\Pi} \\ S \end{bmatrix} = \left[\begin{array}{ccc} 1 & 1 & 1 \\ \hline 1 & -1 & 0 \\ 0 & 1 & -1 \end{array}\right].$$

One can find more about the criteria for choosing valid transformation matrix under certain conditions in [9]. The (x, y) coordinates of the PEs in the obtained SA are determined from

$$\begin{bmatrix} x \\ y \end{bmatrix} = S \cdot \vec{p}^{\,*} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} i+j-\dfrac{2r}{3}-1 \\ j+\dfrac{r}{3} \\ j+k+\dfrac{r}{3}-1 \end{bmatrix} = \begin{bmatrix} i-r-1 \\ 1-k \end{bmatrix},$$

for each $0 \le r \le 2$, $1 \le i \le N_1$, $1 \le j \le N_2$, $1 \le k \le N_3$. The time schedule of the data item indexed by (i, j, k) is given by

$$t = t\left(i+j-\frac{2r}{3}-1, j+\frac{r}{3}, j+k+\frac{r}{3}-1 \right) = i+3j+k-2.$$

Now, for input data items we assume the following periodicity:

$$a\left(i-\frac{2r}{3}+N_1, j+\frac{r}{3}, k+\frac{r}{3} \right) \equiv a\left(i-\frac{2r}{3}, j+\frac{r}{3}, k+\frac{r}{3}+N_3 \right) \equiv a_{ik},$$
$$b\left(i-\frac{2r}{3}, j+\frac{r}{3}+N_2, k+\frac{r}{3} \right) \equiv b\left(i-\frac{2r}{3}, j+\frac{r}{3}, k+\frac{r}{3}+N_3 \right) \equiv b_{kj}.$$

The initial (x, y) positions of input data items are determined from

$$a\left(i+j-\frac{2r}{3}-1, 0, j+k+\frac{r}{3}-1 \right) \longmapsto \begin{bmatrix} x \\ y \end{bmatrix}_a = \begin{bmatrix} 2i+3j+k-r-5 \\ 5-3j-i-2k \end{bmatrix},$$
$$b\left(0, j+\frac{r}{3}, j+k+\frac{r}{3}-1 \right) \longmapsto \begin{bmatrix} x \\ y \end{bmatrix}_b = \begin{bmatrix} 3-3j-k-r \\ 1-k \end{bmatrix},$$
$$c\left(i+j-\frac{2r}{3}-1, j+\frac{r}{3}, 0 \right) \longmapsto \begin{bmatrix} x \\ y \end{bmatrix}_c = \begin{bmatrix} i-r-1 \\ 3j+i-3 \end{bmatrix},$$

for each $0 \le r \le 2$, $1 \le i \le N_1$, $1 \le j \le N_2$, $1 \le k \le N_3$.

The obtained array has the following features:

$$\begin{aligned} n_p &= N_3(N_1+2), \\ t_c &= N_1+3N_2+N_3-4, \\ AT &= N_3(N_1+2)(N_1+3N_2+N_3-4). \end{aligned} \tag{4.3}$$

For $N_1 = N_2 = N_3 = N$, we obtain the same as in (4.2).

According to the previous, we conclude that a hexagonal SA for fault-tolerant matrix multiplication has the following characteristics:

$$
\begin{aligned}
n_p &= N_3(\min\{N_1, N_2\} + 2), \\
t_c &= (3\max\{N_1, N_2\} + \min\{N_1, N_2\} + N_3 - 4), \\
AT &= N_3(\min\{N_1, N_2\} + 2)(3\max\{N_1, N_2\} + \min\{N_1, N_2\} + N_3 - 4).
\end{aligned}
$$

Obviously, for $N_1 = N_2 = N_3 = N$, features of the obtained array are determined by (4.2). For a given problem size, this array has a minimal possible number of PEs needed to perform fault-tolerant matrix multiplication. Compared to the array obtained in [5], the number of PEs is reduced almost three times, space-time complexity, AT, two times, while the total execution time is only slightly increased. The data schedule in this array at the beginning of fault-tolerant matrix multiplication for $N_1 = N_2 = N_3 = 3$ is diagrammed in Figure 2.

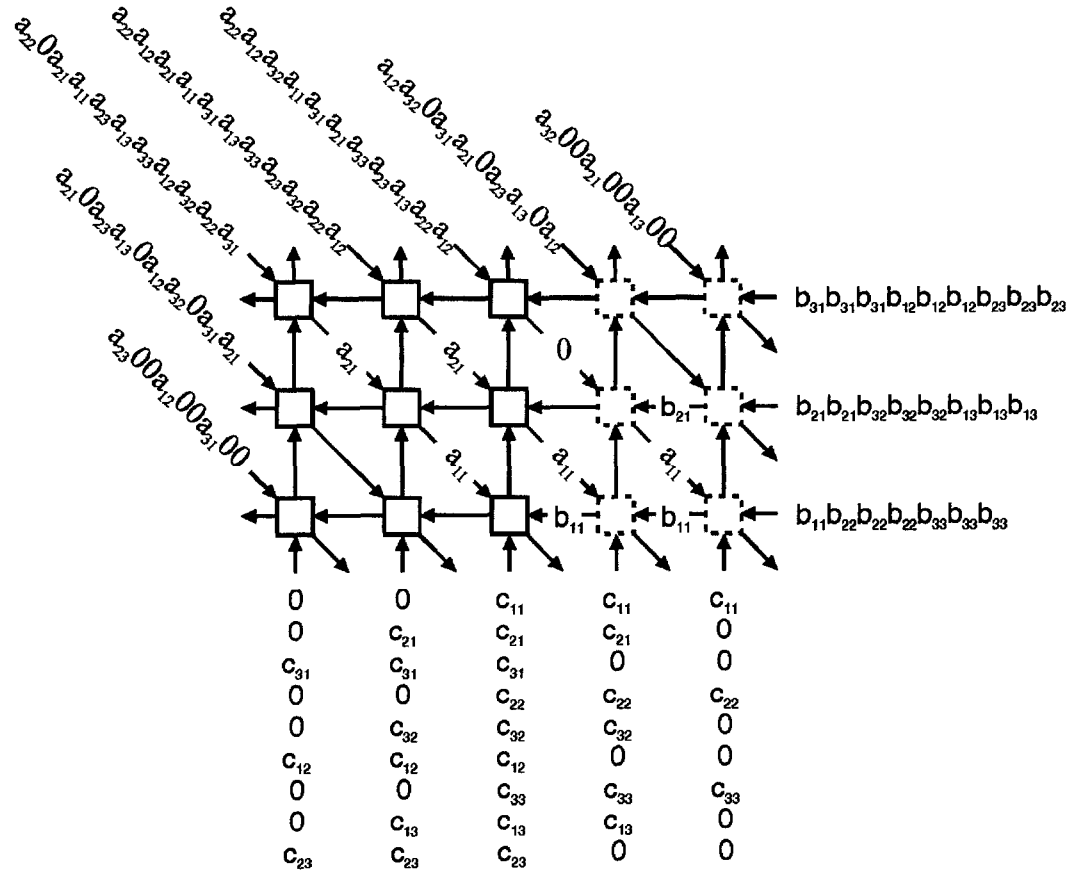

Figure 2. Data flow in the SA synthesized by the described procedure during fault-tolerant matrix multiplication for $N_1 = N_2 = N_3 = 3$.

A detail concerning the voting mechanism is sketched in Figure 3. Figure 3 depicts only the processing elements at the boundary of fault-tolerant SA, since only the final results are subjects in the voting process. Note that each multiplexer (MUX) takes data from three different PEs at three different cycles. The inputs are selected in "round robin" manner starting from input 0. Control signals for all multiplexers are unique. Each voter takes three results to vote. There are $[(N+2)/3] * 3$ multiplexers and $[(N+2)/3]$ voters. By the proposed scheme, a single permanent or temporary fault can be tolerated. A number of multiple fault patterns can be tolerated also, provided that faults do not affect the same element of the resulting matrix.

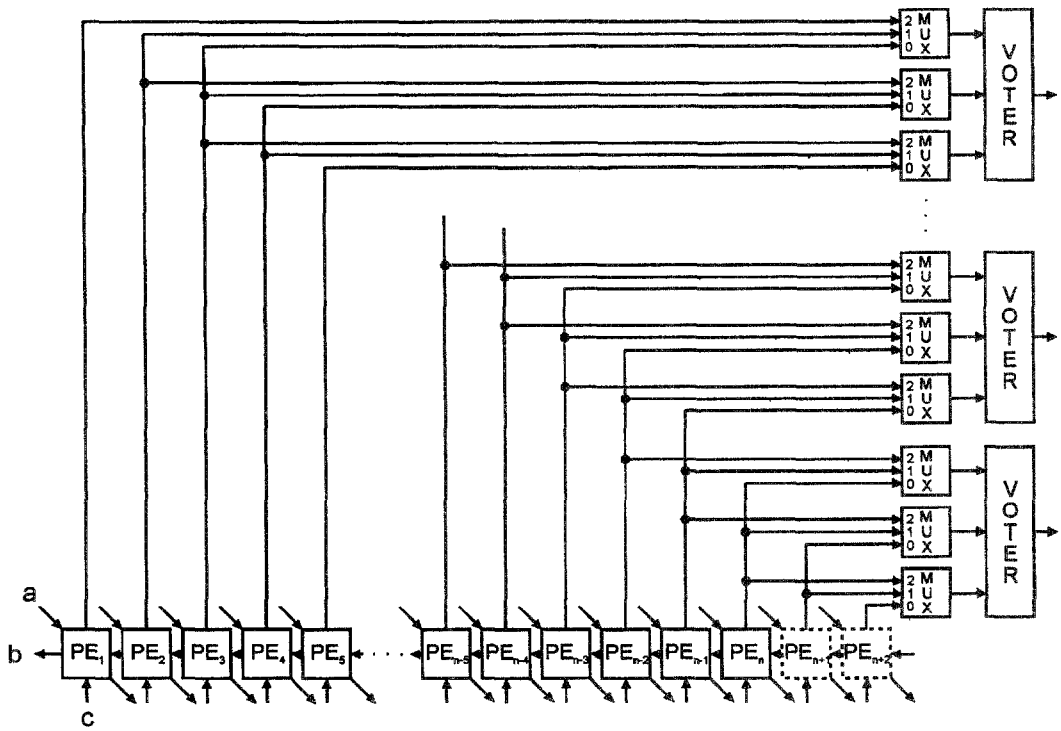

Figure 3. A detail of the voting process.

A better error coverage can be achieved if voting is performed after each computational step. In this case, additional hardware has to be inserted between each row of processing elements. Figure 4 shows the structure of the voting hardware inserted between each two rows of the PEs. Now, there are two levels of multiplexers: $[(N+2)/3]*3$ at the output of PEs, and $N+2$ at the input of PEs. Control signals for all multiplexers in one row are unique and could be pipelined along with the elements of matrix C through the array. Again, the inputs of multiplexers are selected in a "round robin" way, starting from input 0. Each voter takes three results to vote and broadcast the result to three multiplexers. By the proposed scheme, multiple faults occurred on the same resulting element can be tolerated if they appear at different clock cycles.

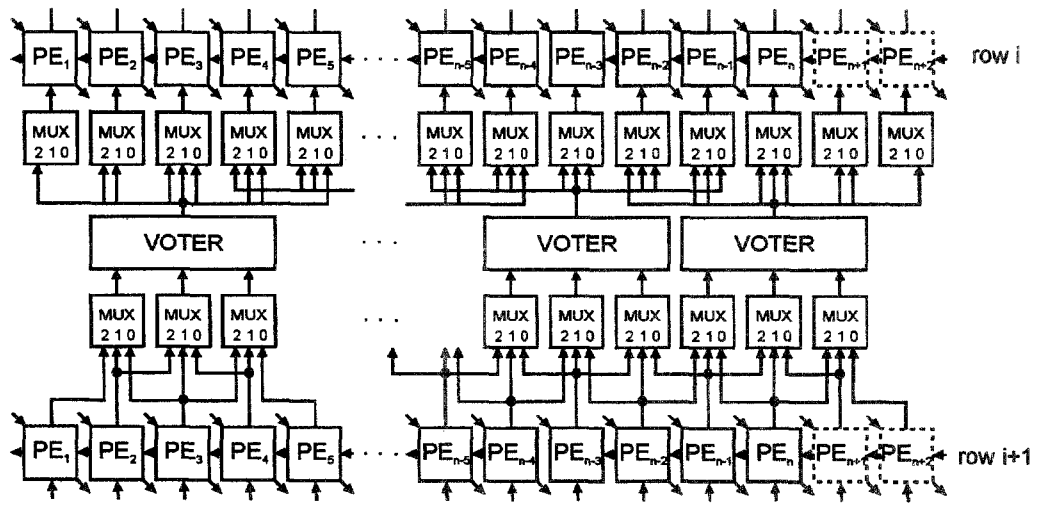

Figure 4. A detail of the voting process when performed between each stage of processing elements.

5. EXPERIMENTAL RESULTS

In order to estimate the error coverage of the proposed method and to compare two hardware solutions, we have performed simulations by inserting errors randomly during the computations for various error rates and dimensions of matrices. Simulation is performed such that the total number of operations of add-multiply type is approximately 10^8 in order to insert an approximately equal number of errors for the same error rates, fp, and various dimensions of matrices, n. Some of the simulation results are presented in Tables 1–3 and graphically in Figures 5–7. For higher error rates, e.g., between 10^{-2} to 10^{-3}, the second hardware solution gives substantially better error coverage, between 86.19% and 99.99%. For error rates $\leq 3.16*10^{-5}$, the second hardware solution provides 100% error coverage. For error rates $\leq 10^{-6}$, both schemes provide 100% error coverage.

Table 1. Simulation results for matrix dimension 50 × 50.

Matrix Dimension, n: 50 × 50 No. of Multiplications: 800 No. of Operations: 300.000.000			Voting Performed at the End		Voting between Each Stage of PEs	
Error Rate (fp)	$-\log(fp)$	No. of Errors	Correct Elements	Error Coverage %	Correct Elements	Error Coverage %
1.00E−02	2	3000211	1309490	65.474	1970386	98.519
3.16E−03	2.5	948085	1883852	94.192	1997003	99.850
1.00E−03	3	300151	1986151	99.307	1999708	99.985
3.16E−03	3.5	94950	1998486	99.924	1999973	99.998
1.00E−04	4	30305	1999847	99.992	1999994	99.999
3.16E−04	4.4	9451	1999991	99.999	2000000	100
1.00E−05	5	3034	1999997	99.999	2000000	100
1.00E−06	6	287	2000000	100	2000000	100
1.00E−07	7	31	2000000	100	2000000	100
1.00E−08	8	5	2000000	100	2000000	100
Total Number of Computed Elements			2000000		2000000	

Table 2. Simulation results for matrix dimension 200 × 200.

Matrix Dimension, n: 200 × 200 No. of Multiplications: 13 No. of Operations: 312.000.000			Voting Performed at the End		Voting between Each Stage of PEs	
Error Rate (fp)	$-\log(fp)$	No. of Errors	Correct Elements	Error Coverage %	Correct Elements	Error Coverage %
1.00E−02	2	3119625	25727	4.947	490017	94.234
3.16E−03	2.5	985209	284395	54.691	516887	99.401
1.00E−03	3	311912	474962	91.338	519695	99.941
3.16E−03	3.5	98451	514436	98.930	519971	99.994
1.00E−04	4	31638	519380	99.880	519995	99.994
3.16E−04	4.5	9908	519937	99.987	520000	99.999
1.00E−05	5	3118	519992	99.998	520000	100
1.00E−06	6	329	520000	100	520000	100
1.00E−07	7	32	520000	100	520000	100
1.00E−08	8	4	520000	100	520000	100
Total Number of Computed Elements			520000		520000	

Table 3. Simulation results for matrix dimension 500 x 500.

Matrix Dimension, n: 500 x 500 No. of Multiplications: 1 No. of Operations: 375.000.000			Voting Performed at the End		Voting between Each Stage of PEs	
Error Rate (fp)	$-\log(fp)$	No. of Errors	Correct Elements	Error Coverage %	Correct Elements	Error Coverage %
1.00E−02	2	3749988	25	0.01	215492	86.196
3.16E−03	2.5	1184690	27302	10.920	246282	98.512
1.00E−03	3	375255	164643	65.857	249594	99.837
3.16E−03	3.5	118708	235431	94.1724	249961	99.984
1.00E−04	4	37542	248296	99.318	249990	99.996
3.16E−04	4.5	11850	249799	99.919	250000	100
1.00E−05	5	3762	249979	99.991	250000	100
1.00E−06	6	370	250000	100	250000	100
1.00E−07	7	38	250000	100	250000	100
1.00E−08	8	4	250000	100	250000	100
Total Number of Computed Elements			250000		250000	

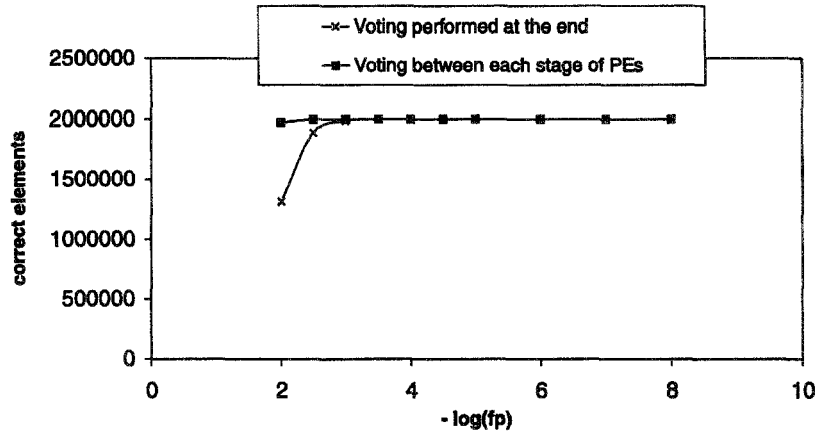

Figure 5. A number of correctly computed elements for various error rates for matrix dimension 50 x 50.

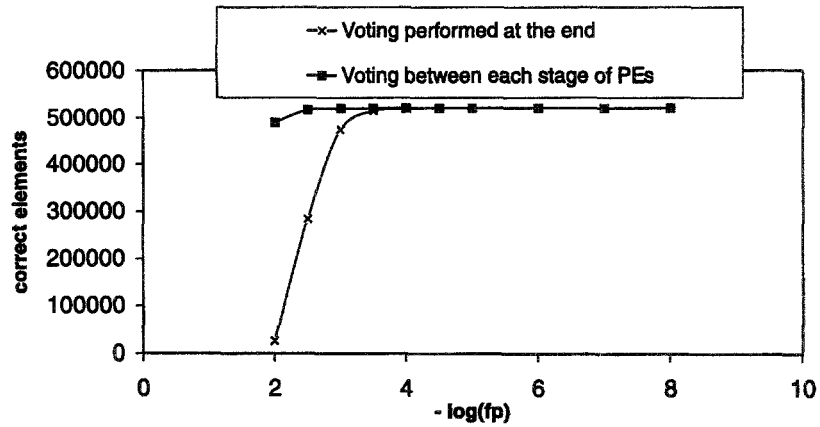

Figure 6. A number of correctly computed elements for various error rates for matrix dimension 200 x 200.

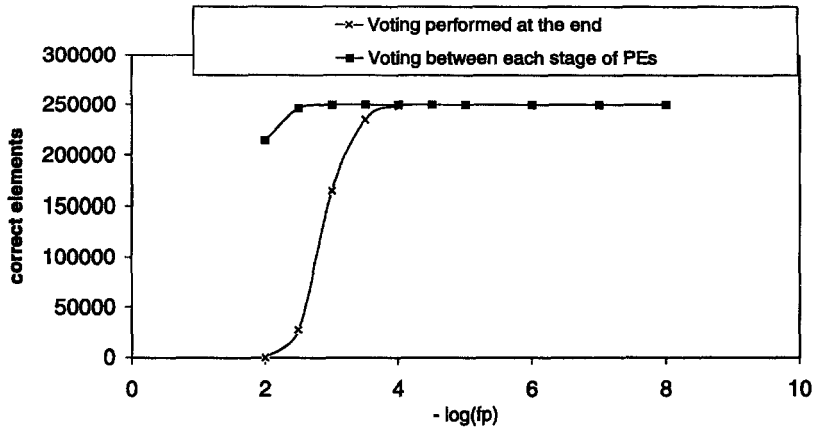

Figure 7. A number of correctly computed elements for various error rates for matrix dimension 500 x 500.

6. CONCLUSION

We have described a method to synthesize an optimal fault-tolerant SA for matrix multiplication with minimal hardware overhead. The array is optimal in the sense of space-time complexity, AT. Its AT measure is almost two times less than that of the array given in [5]. The fault tolerance is achieved through triplicated computation of the same problem instance followed by the majority voting. A single permanent and temporary fault and a number of multiple fault patterns can be tolerated by the proposed scheme. We have described two hardware solutions for the voting process: one when voting is performed at the end of the computation, and the other when it is performed after each computational step. In both cases, fault detection and location are not necessary for fault-tolerance; errors are masked concurrently with normal operation of the systolic array. Experimental results show that a lot of multiple error patterns can be tolerated, also.

REFERENCES

1. K.H. Huang and J.A. Abraham, Algorithm-based fault tolerance for matrix operations, *IEEE Trans. Comput.* **C–33** (6), 518–528, (1984).
2. M.K. Stojčev, E.I. Milovanović and I.Ž. Milovanović, Algorithm based fault-tolerant technique for matrix inversion, In *Proc. of the Int. Conf. Parallel Computing Technologies*, (Edited by N.N. Mirenkov), pp. 375–384, Novosibirsk '91, World Scientific, (1991).
3. M.O. Esonu, A.J. Al-Khalili, S. Hariri and D. Al-Khalili, Fault-tolerant design methodology for systolic array architectures, *IEE Proc. Comput. Digit. Tech.* **141** (1), 17–28, (1994).
4. C.N. Zhang, T.M. Bachtiar and W.K. Chou, Optimal fault-tolerant design approach for VLSI array processors, In *Inter. Conf. Parallel and Distributed Systems*, Taiwan, pp. 348–353, (1994).
5. C.N. Zhang, T.M. Bachtiar and W.K. Chou, Optimal fault-tolerant design approach for VLSI array processors, *IEE Proc. Comput. Digit. Tech.* **144** (1), 15–21, (1997).
6. I.Z. Milentijević, I.Ž. Milovanović, E.I. Milovanović and M.K. Stojčev, The design of optimal planar systolic arrays for matrix multiplication, *Computers Math. Applic.* **33** (6), 17–35, (1997).
7. E.I. Milovanović, I.Z. Milentijević and I.Ž. Milovanović, Designing of processor-time optimal systolic array for matrix multiplication, *Comput. Artificial Intelligence* **16** (1), 1–11, (1997).
8. E.I. Milovanović, G.V. Milovanović, I.Ž. Milovanović and D. Milosavljević, Designing hexagonal systolic array by composite mappings, *Facta Universitatis (Niš), Ser. Math. Inform.* **12**, 283–296, (1997).
9. T.I. Tokić, I.Ž. Milovanović, D.M. Randjelović and E.I. Milovanović, Determining VLSI array size for one class of nested loop algorithms, In *Advances in Computer and Information Sciences '98*, (Edited by U. Gudukbay), pp. 389–396, IOS Press, (1998).
10. C.N. Zhang, Systematic design of systolic arrays for computing multiple problem instances, *Microelectronics Journal* **23**, 543–553, (1992).
11. J.-J. Wang and C.-W. Jen, Redundancy design for a fault tolerant systolic array, *IEE Proceedings* **137**, **Pt. E** (3), 218–226, (1990).
12. C.N. Zhang, J.H. Weston and Y.F. Yan, Determining objective functions in systolic array designs, *IEEE Trans. on VLSI Systems* **2** (3), 357–360, (1994).

13. J.C. Tsay and P.Y. Zhang, Space-optimal linear arrays for algorithms with linear schedules, *IEEE Trans. Comp.* **5**, 683–694, (1995).
14. C. Langauer, A view of systolic design, In *Proc. International Conference Parallel Computing Technologies*, (Edited by N.N. Mirenkov), pp. 32–46, Novosibirsk '91, World Scientific, Singapore, (1991).
15. D.J. Evans and C.R. Wan, Massive parallel processing for matrix multiplication: A systolic approach, In *Highly Parallel Computations: Algorithms and Applications*, (Edited by M.P. Bekakos), pp. 139–173, WIT Press, Southampton, Boston, MA, (2001).